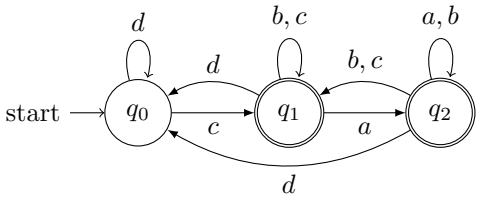
  - $q_1 \xrightarrow{\epsilon} q_0 \xrightarrow{d} q_0$ is replaced by $q_1 \xrightarrow{d} q_0$;
  - $q_2 \xrightarrow{\epsilon} q_1 \xrightarrow{a} q_2$ is replaced by $q_2 \xrightarrow{a} q_2$;
  - $q_2 \xrightarrow{\epsilon} q_1 \xrightarrow{b} q_1$ is replaced by $q_2 \xrightarrow{b} q_1$;
  - $q_2 \xrightarrow{\epsilon} q_1 \xrightarrow{\epsilon} q_0 \xrightarrow{d} q_0$ is replaced by $q_2 \xrightarrow{d} q_0$; and
  - $q_2 \xrightarrow{\epsilon} q_1 \xrightarrow{\epsilon} q_0 \xrightarrow{c} q_1$ is replaced by $q_2 \xrightarrow{c} q_1$.

- For epsilon transition chains ending at a final state, we have the following:
  - $q_2 \xrightarrow{\epsilon} q_1$, so state q_2 becomes a final state.

Adding these transitions and final states produces our nondeterministic finite automaton without epsilon transitions:



In our next procedure, we will see a way to "simulate" nondeterminism in a deterministic finite automaton. Recall that, in a nondeterministic finite automaton, the transition function maps state/symbol pairs to an element of $\mathcal{P}(Q)$. We can get around the issue of having multiple transitions from one state on the same symbol not by changing our transitions, but by changing our set of states: we simply need to create one state corresponding to each element of $\mathcal{P}(Q)$!

Theorem 19. *Given a nondeterministic finite automaton $\mathcal{N}$, we can convert it to a deterministic finite automaton $\mathcal{N}'$.*

Proof. Let $\mathcal{N} = (Q, \Sigma, \delta, q_0, F)$ be a nondeterministic finite automaton. (We assume that $\mathcal{N}$ contains no epsilon transitions; if it does, then use the construction of Theorem 17 to remove the epsilon transitions.) We will construct a deterministic finite automaton $\mathcal{N}' = (Q', \Sigma, \delta', q_0', F')$ in the following way:

1. Take $Q' = \mathcal{P}(Q)$; that is, each state of $\mathcal{N}'$ corresponds to a subset of states of $\mathcal{N}$. Note that our deterministic finite automaton may not need to use all of these states; usually, we omit any inaccessible states to make our diagram easier to follow.
2. For each $q' \in Q'$ and $a \in \Sigma$, take $\delta'(q', a) = \{q \in Q \mid q \in \delta(s, a) \text{ for some } s \in q'\}$.

   (This is perhaps the most difficult step of the construction. Remember that each state q' of $\mathcal{N}'$ corresponds to a subset of states of $\mathcal{N}$. Thus, when $\mathcal{N}'$ reads a symbol a in state q', the transition function δ' takes us to the state corresponding to the subset of states q of $\mathcal{N}$ that we would have transitioned to upon reading a in some state s of $\mathcal{N}$, where s is in the subset corresponding to q'.)
3. Take $q_0' = \{q_0\}$; that is, the initial state of $\mathcal{N}'$ corresponds to the subset containing only the initial state of $\mathcal{N}$.
4. Take $F' = \{q' \in Q' \mid q' \text{ corresponds to a subset containing at least one final state of } \mathcal{N}\}$. In this way, $\mathcal{N}'$ accepts only if $\mathcal{N}$ would be in a final state at the same point in its computation.

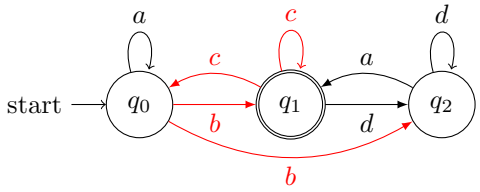
In this way, we have constructed a deterministic finite automaton recognizing the same language as the original finite automaton. □

The procedure allowing us to convert from nondeterministic to deterministic finite automata is known as the *subset construction*, because each state of our deterministic finite automaton corresponds to a subset of states from the original nondeterministic finite automaton.

For step 2 of the conversion process, we may obtain the transition function of our deterministic finite automaton $\mathcal{N}'$ using a tabular method. With this method, we perform the following steps:

1. Construct a table where the rows are the states of $\mathcal{N}$ and the columns are the symbols of the alphabet Σ.
2. For each state q_i and symbol a, write the set of states mapped to by $\delta(q_i, a)$ in the corresponding row/column entry.
3. After all entries are filled, take all sets of states listed in the table that don't yet have their own row, and create a new row corresponding to that set of states.
4. Repeat steps 2 and 3 until no new rows can be added to the table.

Example 20. Consider the following nondeterministic finite automaton, with nondeterministic transitions from a state highlighted in red:



We will use our tabular construction method to obtain the transition function of our desired deterministic finite automaton. Our initial table looks like the following:

	a	b	c	d
q_0				
q_1				
q_2				

We fill in the initial table entries by consulting the transition function of $\mathcal{N}$, where — denotes no transition:

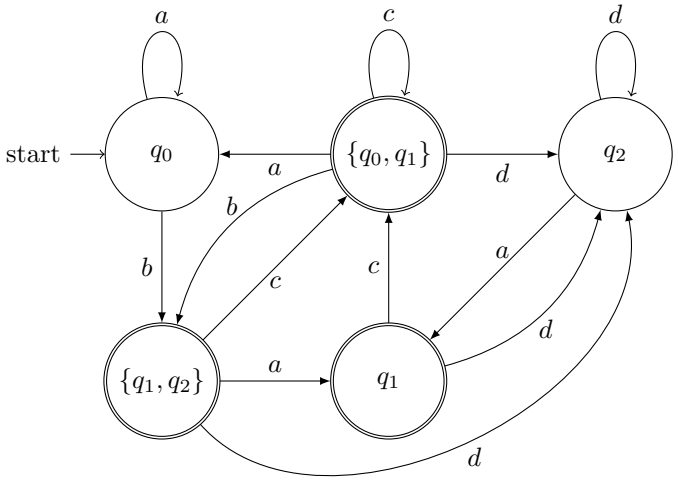
	a	b	c	d
q_0	q_0	$\{q_1, q_2\}$	—	—
q_1	—	—	$\{q_0, q_1\}$	q_2
q_2	q_1	—	—	q_2

Note that there are now two entries in our table without corresponding rows: $\{q_0, q_1\}$ and $\{q_1, q_2\}$. We proceed to add these entries as rows to our table and we fill in the entries for these new rows:

	a	b	c	d
q_0	q_0	$\{q_1, q_2\}$	—	—
q_1	—	—	$\{q_0, q_1\}$	q_2
q_2	q_1	—	—	q_2
$\{q_0, q_1\}$	q_0	$\{q_1, q_2\}$	$\{q_0, q_1\}$	q_2
$\{q_1, q_2\}$	q_1	—	$\{q_0, q_1\}$	q_2

After filling in these new entries, we find that all entries now have corresponding rows, so our table construction is complete. We can now use this table to construct our deterministic finite automaton! Each row of the table corresponds to an accessible state of our deterministic finite automaton, and the table itself specifies our transition function.

Our resultant deterministic finite automaton is the following:



Note that we don't need to come up with procedures for the other direction of conversion: a deterministic finite automaton is already a nondeterministic finite automaton that doesn't use nondeterminism, and a nondeterministic finite automaton is a "nondeterministic finite automaton with epsilon transitions" that doesn't use any epsilon transitions.

Since we now have methods to convert between all three of our models, we can conclude that they are all equivalent in terms of recognition power.

2.5 Closure Properties

Closure properties are an important consideration when we discuss any model of computation, since it allows us to determine whether we can apply certain operations to words or languages while still allowing the model to accept or recognize the result.

We say that a set S is *closed* under an operation $\circ$ if, given any two elements $a, b \in S$, we have that $a \circ b \in S$ as well. You might be familiar with the notion of closure from elsewhere in mathematics: for example, the set of integers is closed under the operations of addition, subtraction, and multiplication, since for all integers a and b, we know that $a + b$, $a - b$, and $a \times b$ are integers. On the other hand, the set of integers is not closed under the operation of division, since (for example) $1, 2 \in \mathbb{Z}$ but $1/2 \notin \mathbb{Z}$.

We can prove all kinds of closure results for languages recognized by finite automata, but here we will focus on three results in particular. In each result, we will follow the same general style of proof to show closure under the specified operation $\circ$: given two finite automata $\mathcal{M}$ and $\mathcal{N}$ recognizing languages $L(\mathcal{M})$ and $L(\mathcal{N})$, we will construct a new finite automaton recognizing the language $L(\mathcal{M}) \circ L(\mathcal{N})$.

We begin by considering the regular operation of union.

Theorem 21. *The class ϵ-NFA is closed under the operation of union.*

Proof. Suppose we are given two nondeterministic finite automata with epsilon transitions, denoted $\mathcal{A} = (Q_{\mathcal{A}}, \Sigma, \delta_{\mathcal{A}}, q_{0_{\mathcal{A}}}, F_{\mathcal{A}})$ and $\mathcal{B} = (Q_{\mathcal{B}}, \Sigma, \delta_{\mathcal{B}}, q_{0_{\mathcal{B}}}, F_{\mathcal{B}})$. We construct a finite automaton $\mathcal{C}$ recognizing the language $L(\mathcal{A}) \cup L(\mathcal{B})$ in the following way:

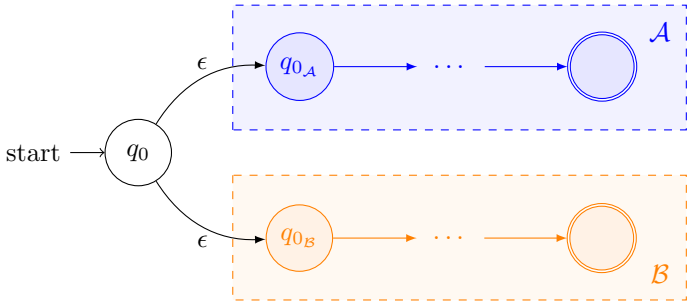
- Take $Q_{\mathcal{C}} = Q_{\mathcal{A}} \cup Q_{\mathcal{B}} \cup \{q_0\}$.
- Take $q_{0_{\mathcal{C}}} = q_0$.

- Take $F_{\mathcal{C}} = F_{\mathcal{A}} \cup F_{\mathcal{B}}$.
- Define $\delta_{\mathcal{C}}$ such that, for all $q \in Q_{\mathcal{C}}$ and for all $a \in \Sigma \cup \{\epsilon\}$,

$$\delta_{\mathcal{C}}(q,a) = \begin{cases} \delta_{\mathcal{A}}(q,a) & \text{if } q \in Q_{\mathcal{A}}; \\ \delta_{\mathcal{B}}(q,a) & \text{if } q \in Q_{\mathcal{B}}; \text{ and} \\ \{q_{0_{\mathcal{A}}}, q_{0_{\mathcal{B}}}\} & \text{if } q = q_0 \text{ and } a = \epsilon. \end{cases}$$

□

Diagrammatically, the "union" finite automaton $\mathcal{C}$ looks like the following:



Next, we consider the operation of concatenation.

Theorem 22. *The class ϵ-NFA is closed under the operation of concatenation.*

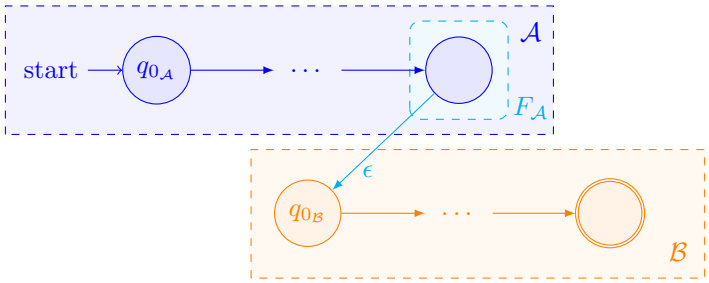
Proof. Suppose we are given two nondeterministic finite automata with epsilon transitions, denoted $\mathcal{A} = (Q_{\mathcal{A}}, \Sigma, \delta_{\mathcal{A}}, q_{0_{\mathcal{A}}}, F_{\mathcal{A}})$ and $\mathcal{B} = (Q_{\mathcal{B}}, \Sigma, \delta_{\mathcal{B}}, q_{0_{\mathcal{B}}}, F_{\mathcal{B}})$. We construct a finite automaton $\mathcal{C}$ recognizing the language $L(\mathcal{A})L(\mathcal{B})$ in the following way:

- Take $Q_{\mathcal{C}} = Q_{\mathcal{A}} \cup Q_{\mathcal{B}}$.
- Take $q_{0_{\mathcal{C}}} = q_{0_{\mathcal{A}}}$.
- Take $F_{\mathcal{C}} = F_{\mathcal{B}}$.
- Define $\delta_{\mathcal{C}}$ such that, for all $q \in Q_{\mathcal{C}}$ and for all $a \in \Sigma \cup \{\epsilon\}$,

$$\delta_{\mathcal{C}}(q,a) = \begin{cases} \delta_{\mathcal{A}}(q,a) & \text{if } q \in Q_{\mathcal{A}} \text{ and } q \notin F_{\mathcal{A}}; \\ \delta_{\mathcal{A}}(q,a) & \text{if } q \in F_{\mathcal{A}} \text{ and } a \neq \epsilon; \\ \delta_{\mathcal{A}}(q,a) \cup \{q_{0_{\mathcal{B}}}\} & \text{if } q \in F_{\mathcal{A}} \text{ and } a = \epsilon; \text{ and} \\ \delta_{\mathcal{B}}(q,a) & \text{if } q \in Q_{\mathcal{B}}. \end{cases}$$

□

Diagrammatically, the "concatenation" finite automaton $\mathcal{C}$ looks like the following:



For this last result, pertaining to the Kleene star, we are considering just one finite automaton instead of two, but the same general style of proof applies.

Theorem 23. *The class ϵ-NFA is closed under the operation of Kleene star.*

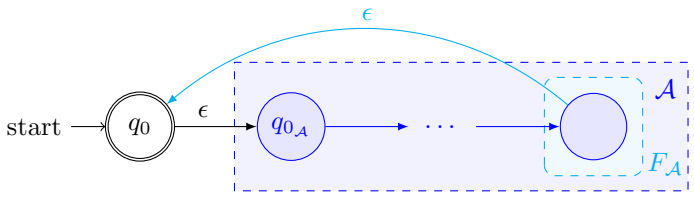
Proof. Suppose we are given a nondeterministic finite automaton with epsilon transitions, denoted $\mathcal{A} = (Q_{\mathcal{A}}, \Sigma, \delta_{\mathcal{A}}, q_{0_{\mathcal{A}}}, F_{\mathcal{A}})$. We construct a finite automaton $\mathcal{A}'$ recognizing the language $L(\mathcal{A})^*$ in the following way:

- Take $Q_{\mathcal{A}'} = Q_{\mathcal{A}} \cup \{q_0\}$.
- Take $q_{0_{\mathcal{A}'}} = q_0$.
- Take $F_{\mathcal{A}'} = \{q_0\}$.
- Define $\delta_{\mathcal{A}'}$ such that, for all $q \in Q_{\mathcal{A}'}$ and for all $a \in \Sigma \cup \{\epsilon\}$,

$$\delta_{\mathcal{A}'}(q,a) = \begin{cases} \delta_{\mathcal{A}}(q,a) & \text{if } q \in Q_{\mathcal{A}} \text{ and } q \notin F_{\mathcal{A}}; \\ \delta_{\mathcal{A}}(q,a) & \text{if } q \in F_{\mathcal{A}} \text{ and } a \neq \epsilon; \\ \delta_{\mathcal{A}}(q,a) \cup \{q_0\} & \text{if } q \in F_{\mathcal{A}} \text{ and } a = \epsilon; \text{ and} \\ \{q_{0_{\mathcal{A}}}\} & \text{if } q = q_0 \text{ and } a = \epsilon. \end{cases}$$

□

Diagrammatically, the "Kleene star" finite automaton $\mathcal{A}'$ looks like the following:



Since we know how to convert between all of our models, we naturally get that the classes NFA and DFA are also closed under each of the union, concatenation, and Kleene star operations.

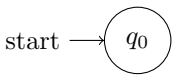
Recall, though, that we referred to each of these operations by a special name: the regular operations. Moreover, we said that any language that can be constructed using these regular operations was a regular language. Since all of our finite automaton models are closed under the regular operations, we arrive at the first truly exciting result of this lecture: finite automata recognize *exactly* the class of regular languages.

Theorem 24. *A language A is regular if and only if there exists a deterministic finite automaton $\mathcal{M}$ such that $L(\mathcal{M}) = A$.*

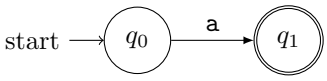
Proof. ($\Rightarrow$): To prove this direction of the statement, we need to construct a deterministic finite automaton for each of the five "basic" regular languages specified in Definition 3.

We already know how to construct a finite automaton for each of the union, concatenation, and Kleene star languages from the proofs of Theorems 21, 22, and 23; we just need to take the extra step of converting the finite automata from each of those proofs to their deterministic equivalent. Thus, it suffices to construct deterministic finite automata for the remaining regular languages: the empty language $\emptyset$ and the singleton language $\{a\}$ for each $a \in \Sigma$.

The empty language $\emptyset$ is recognized by the following deterministic finite automaton:



For all $a \in \Sigma$, the singleton language is recognized by the following deterministic finite automaton:



($\Leftarrow$): This direction of the statement is trivially true, by Definition 10. □

3 Regular Expressions

If you frequently use a Unix-based system with a terminal, you may be familiar with utilities such as `grep`, which searches an input text file for lines that match a specified format. For example, on my computer, I can search the dictionary file (`/usr/share/dict/words`) for all words that contain "theory":

```
taylor@SmithBook:~> grep theory /usr/share/dict/words
countertheory
theory
theoryless
theorymonger
```

But, to be fair, doing something like that is a bit overkill when one could just open the file in a text editor and use the Find tool to search for the word "theory". Where `grep` really shines is when we need to search for text matching a *pattern*, like so:

```
taylor@SmithBook:~> grep ^u.*ity$ /usr/share/dict/words
ubiquity
ultimity
ultrafilterability
...
usability
utility
utterability
uxoriality
```

In this example, I searched for all words in `/usr/share/dict/words` that began with a `u` and ended with `ity`, such as `university`. The `ubiquity` of this pattern in the English language is evident:

```
taylor@SmithBook:~> grep ^u.*ity$ /usr/share/dict/words | wc -l
     235
```

Utilities like `grep` use patterns to perform fast searches in text files, and we can formalize the notion of a pattern in terms of our familiar regular operations. This formalization is known as a *regular expression*.

3.1 Definition

To define regular expressions, let us recall our first characterization of regular languages from Definition 3. We stated that a language was regular if we could represent it in terms of five "components": the empty language, $\emptyset$; the singleton language $\{a\}$ for all $a \in \Sigma$; the union operation; the concatenation operation; and the Kleene star operation. To define regular expressions, we really don't need to make any big changes to this characterization, apart from thinking about symbols instead of languages!

Definition 25 (Regular expression)**.** Let Σ be an alphabet. The class of regular expressions is defined inductively as follows:

1. $\boldsymbol{r} = \emptyset$ is a regular expression;
2. $\boldsymbol{r} = \epsilon$ is a regular expression;
3. For each $a \in \Sigma$, $\boldsymbol{r} = a$ is a regular expression;
4. For regular expressions $\boldsymbol{r}_1$ and $\boldsymbol{r}_2$, $\boldsymbol{r} = \boldsymbol{r}_1 + \boldsymbol{r}_2$ is a regular expression;
5. For regular expressions $\boldsymbol{r}_1$ and $\boldsymbol{r}_2$, $\boldsymbol{r} = \boldsymbol{r}_1\boldsymbol{r}_2$ is a regular expression; and
6. For a regular expression $\boldsymbol{r}$, $\boldsymbol{r}^*$ is a regular expression.

Every regular expression *represents* a language, and we denote the language represented by a regular expression $\boldsymbol{r}$ by $L(\boldsymbol{r})$. Note that each of the six "base" regular expressions correspond to their own language. If $\boldsymbol{r} = \emptyset$, then $L(\boldsymbol{r}) = \emptyset$. Likewise, if $\boldsymbol{r} = \epsilon$, then $L(\boldsymbol{r}) = \{\epsilon\}$, and if $\boldsymbol{r} = a$, then $L(\boldsymbol{r}) = \{a\}$. The remaining three regular expressions correspond exactly to the regular operations of union, concatenation, and Kleene star. We will denote the class of languages represented by some regular expression by RE.

Example 26. Let $\Sigma = \{\mathtt{a}, \mathtt{b}\}$, and consider the language $L_{\text{odda}} = \{w \mid w \text{ contains an odd number of } \mathtt{a}\text{s}\}$. This language is represented by the regular expression $\boldsymbol{r}_{\text{odda}} = \mathtt{b}^*(\mathtt{ab}^*\mathtt{ab}^*)^*\mathtt{ab}^*$. The first component of $\boldsymbol{r}$, $\mathtt{b}^*$, recognizes zero or more leading $\mathtt{b}$s. The middle component, $(\mathtt{ab}^*\mathtt{ab}^*)^*$, recognizes zero or more pairs of $\mathtt{a}$s, where each $\mathtt{a}$ is followed by zero or more $\mathtt{b}$s. The last component, $\mathtt{ab}^*$, recognizes an additional $\mathtt{a}$ to ensure the total number of $\mathtt{a}$s is odd, followed by zero or more $\mathtt{b}$s.

Note that this regular expression is not unique; the same language is recognized by the regular expression $\boldsymbol{r}'_{\text{odda}} = \mathtt{b}^*\mathtt{ab}^*(\mathtt{ab}^*\mathtt{ab}^*)^*$.

Example 27. Consider the regular expression $\boldsymbol{r} = (\mathtt{a} + \mathtt{b})^*\mathtt{b}$ over the alphabet $\Sigma = \{\mathtt{a}, \mathtt{b}\}$. We can "decompose" the language represented by $\boldsymbol{r}$ in the following way:

$$\begin{aligned} L(\boldsymbol{r}) &= L((\mathtt{a}+\mathtt{b})^*\mathtt{b}) \\ &= L(\mathtt{a}+\mathtt{b})^* L(\mathtt{b}) \\ &= (L(\mathtt{a}) \cup L(\mathtt{b}))^* L(\mathtt{b}) \\ &= (\{\mathtt{a}\} \cup \{\mathtt{b}\})^* \{\mathtt{b}\} \\ &= \{\mathtt{a}, \mathtt{b}\}^* \{\mathtt{b}\}. \end{aligned}$$

Therefore, $\boldsymbol{r}$ represents the language $L = \{w \mid w \text{ ends with } \mathtt{b}\}$.

Example 28. The empty word ϵ and the empty language $\emptyset$ operate a little differently than other words and languages in regular expressions.

- For the empty word ϵ and any regular expression $\boldsymbol{r}$, we have that $\boldsymbol{r}\epsilon = \boldsymbol{r}$ but $\boldsymbol{r} + \epsilon \neq \boldsymbol{r}$ in general. We also have that $\epsilon^* = \{\epsilon\}$.
- For the empty language $\emptyset$ and any regular expression $\boldsymbol{r}$, we have that $\boldsymbol{r} + \emptyset = \boldsymbol{r}$ but $\boldsymbol{r}\emptyset = \emptyset$. We also have that $\emptyset^* = \{\epsilon\}$.

Just like how mathematics has an order of operations, regular expressions abide by their own order of precedence. The Kleene star is applied first, followed by concatenation, and then union. We can modify the order in which operations are applied by adding parentheses to a regular expression, which do not affect the language represented by that regular expression.

Example 29. Consider the regular expressions $\boldsymbol{r}_1 = \mathtt{0} + \mathtt{1}^*\mathtt{10} + \mathtt{1}^*$ and $\boldsymbol{r}_2 = (\mathtt{0} + \mathtt{1})^*\mathtt{1}(\mathtt{0} + \mathtt{1})^*$. Clearly, the only visual difference between $\boldsymbol{r}_1$ and $\boldsymbol{r}_2$ is the addition of parentheses. However, the languages represented by $\boldsymbol{r}_1$ and $\boldsymbol{r}_2$ are quite different:

- The expression $\boldsymbol{r}_1$ represents the language containing (i) the word `0`, (ii) all words consisting of at least one `1` with one `0` at the end, and (iii) all words consisting of zero or more `1`s.
- The expression $\boldsymbol{r}_2$ represents the language consisting of all words that contain at least one `1`.

We may additionally define some shorthand notation to make our regular expressions look nicer, though strictly speaking, this notation is not "official". We may choose to write $\boldsymbol{r}^+ = \boldsymbol{r}\boldsymbol{r}^* = \boldsymbol{r}^*\boldsymbol{r}$, which is sometimes called the *Kleene plus* operation. Similarly, we may use exponents to denote iterated concatenation; that is, $\boldsymbol{r}^k$ denotes $\boldsymbol{r}$ concatenated with itself k times.

You may now be wondering why our earlier command line examples used pattern-matching symbols like `^`, `.`, and `$`, when Definition 25 didn't define any of those symbols. This is because our definition of a regular expression is the purely-theoretic definition, and as the name suggests, it is meant to coincide exactly with our definition of a regular language. It is therefore different from the practical implementation of "regular expressions", where we can use special symbols to indicate the start or end of a word, match any symbol instead of one specific symbol, and make back-references among other things. The literature refers to these practical implementations as *extended regular expressions*, and these expressions can, in fact, represent more than just the class of regular languages. Thus, in the context of this lecture, when we refer to a regular expression we will be following Definition 25.

3.2 RE = DFA

At this point, we now have three ways to represent a regular language: by using regular operations, by constructing a finite automaton, and by writing a regular expression. In some cases, it's easier for us to represent a regular language using a regular expression, while in other cases it might be easier for us to construct a finite automaton to recognize the language. However, is it always the case that, if we can do one, we can also do the other?

We already know from the last section that the first two representations are equivalent in terms of their descriptive power, so it would be nice to show that our new method of representation is similarly equivalent. By doing so, we can translate between all three representations. This brings us to the next exciting result of the lecture.

Theorem 30. *A language A is regular if and only if there exists a regular expression $\boldsymbol{r}$ such that $L(\boldsymbol{r}) = A$.*

Proof. ($\Rightarrow$): To prove this direction of the statement, we will take a deterministic finite automaton recognizing the language A, and then convert the finite automaton to a regular expression. We will use a *state elimination algorithm* to perform this conversion.

Note that, for this proof only, we will assume that the transitions of our finite automaton can be labelled by regular expressions and not just symbols.

Suppose that we are given a deterministic finite automaton $\mathcal{M}$ such that $L(\mathcal{M}) = A$. Further suppose, without loss of generality, that there exists at most one transition between any two states of $\mathcal{M}$; we can make this assumption since multiple transitions between two states on symbols $a_1, \ldots, a_n$ can be replaced by the single transition on the regular expression $a_1 + \cdots + a_n$.

If $\mathcal{M}$ contains no accepting state, then $A = \emptyset$ and we are done. Otherwise, if $\mathcal{M}$ contains multiple accepting states, convert them to non-accepting states and add epsilon transitions from each formerly-accepting state to a new single accepting state. If the initial state is also an accepting state, make a similar change to the initial state.